A Multi Interface Grid Discovery System

A. Ali[1], A. Anjum[1,2], J. Bunn[3], F. Khan[1], R.McClatchey[2], H. Newman[3], C. Steenberg[3], M. Thomas[3], Ian Willers[4]

[1]National University of Sciences and Technology, Rawalpindi, Pakistan
Email:{Faisal.khan,Arshad.Ali}@niit.edu.pk
[2]CCS Research Centre, Univ. of West of England, Frenchay, Bristol BS16 1QY, UK
Email:{Richard.McClatchey,Ashiq.Anjum}@cern.ch
[3]California Institute of Technology, United States
Email:{ julian.bunn@caltech.edu}{newman, conrad, thomas@hep.caltech.edu}
[4]European Organization for Nuclear Research, Geneva, Switzerland
Email: Ian.Willers@cern.ch

Abstract

Discovery Systems (DS) can be considered as entry points for global loosely coupled distributed systems. An efficient Discovery System in essence increases the performance, reliability and decision making capability of the whole distributed system particularly if the complexity is large as in Grid systems. With the rapid increase in scale of distributed applications, existing solutions for discovery systems are fast becoming either obsolete or incapable of handling such complexity. They are particularly ineffective when handling service lifetimes and providing up-to-date information, poor at enabling dynamic service access and they can also impose unwanted restrictions on interfaces to widely available information repositories. In this paper we present essential design characteristics, an implementation and a performance analysis for a discovery system capable of overcoming these deficiencies in large, globally distributed environments.

1. Introduction

Grid computing promises to provide solutions to complex scientific and industrial problems. This is achieved by pooling together resources (CPU, storage and networks) from individual or multiple institutes – referred to as ‘virtual organizations’. With the increasingly widespread adoption of the Grid paradigm of computing, there is an unprecedented increase in the number of accessible elements (i.e. CPUs, storage elements, networks etc) connected to the Grid. However this increase comes with a price in terms of the management and coordinated use of these resources, exposed as (web) services. The nature of these services varies from highly stable to highly volatile. Services may appear or disappear: new services can become part of the Grid and older services can be withdrawn from the Grid. Moreover, the location of the services cannot be foreseen. It is therefore important to have a class of services in the distributed system that provides scalable and robust registration and discovery services.

Discovery services or information services enable service providers, scientists and applications to query for services and to retrieve up-to-date information on demand on their location and interfaces. An information service can also act as a facilitator for interaction between other Grid components; it is impossible for a service to interact with another without knowledge of its location. One such example of an interaction facilitated through the discovery service is that of a job scheduler service, in which an instance of a scheduler running on one site can interact with schedulers running on other sites to enable selection of the optimal execution site. Another example is an interaction among different replica location services for choosing the best copy of data – where “best” could, for example, mean ease of access or speed of access. There are many such scenarios where discovery services play major roles in enabling communication among various components of the Grid. Figure 1 shows a skeleton of a generic discovery service.

Most existing service discovery solutions are not suitable for the dynamic and distributed nature of the Grid, for several reasons. The main problems with current implementations are: lack of peer-to-peer discovery, lack of compliance with standards, and an absence of dynamic discovery capability for the renewal of service and multiple interfaces. Another important consideration for designing DSs is to build in fault tolerance mechanisms: this is especially important in a distributed system where the failure of one component can have a domino effect on others.

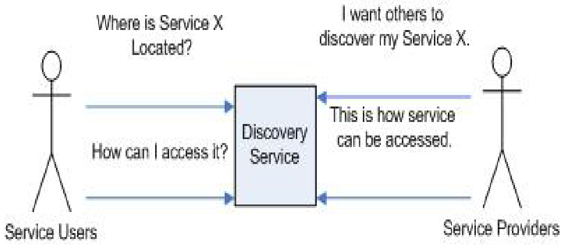

Where is Service X Located?
I want others to discover my Service X.
How can I access it?
Discovery Service
This is how service can be accessed.
Service Users
Service Providers

Figure 1. Service Discovery Skeleton

The DS solution that we have developed is available as part of the Clarens web services framework [1], a Grid portal for scientists which enable them to run Grid-based analyses on high energy physics data. Clarens is being used by different communities under the OSG (Open Science Grid) [3] and by the NVO (National Virtual Observatory) [23].

In the following section we identify the essential characteristics of a Grid information system. The architecture and implementation details of our DS are described in sections 3 and 5 respectively. We discuss the existing solutions based on these features in section 4. We then discuss existing and possible scenarios in which the DS can be utilized. A performance analysis of the DS is presented in section 7. We conclude the paper by describing possible avenues for future work.

2. Design Characteristics

In this section we list design requirements for building a dynamic and distributed system, which can help to register and invoke Grid services virtually instantaneously.

The convergence of Grid and Peer to Peer (P2P) computing offers many advantages [3], [4]. The main advantage is scalability. Applications like Napster [5], Gnutella [6] and SETI@HOME [7] are examples of how P2P systems can exploit a large number of loosely interconnected nodes. Moreover there are other characteristics of P2P systems which can be exploited when implementing a discovery service. One such characteristic is content search. Content search allows P2P networks to keep a catalogue of distributed resources (files). The scale of P2P networks makes searching this catalogue a complex problem since the nodes can join and leave the network effectively at random. The churn rate - which is the rate at which nodes leave and join the network - is much higher in such applications as compared to that of Grid. The distributed search capabilities of P2P file sharing applications can be utilized or studied for improved performance in the Grid environment.

We provide support for multiple plug-ins of different components depending upon the requirements of the Grid users. This is important because of the rapid changes in existing components due to evolving Grid standards. Providing a plug-in API supports flexibility in incorporating new components or in improving existing ones without affecting the overall functionality of the system. The main reason for opting for a multi plug-in approach is that there are multiple applications which exhibit the same or very similar (discovery) functionality but which are used by different communities. UDDI [8], ebXML, relational databases and LDAP can all be used as registries for storing service information. Applications like MonALISA, RGMA, MDS4, GridICE [24] etc. can be used for replication of discovery information among the different discovery nodes. These applications are in use by different Grid communities. For example EGEE is using RGMA while OSG is using MonALISA. The pluggable approach enables us to broaden the usage of the discovery service across multiple communities and also to take advantage of different heterogeneous products, many of which exhibit complementary strengths and weaknesses.

Another important design consideration is the service information lifetime. There can be many cases when information on a service is available within the discovery service but the service itself is down. These ‘stale’ entries can lead to a degradation of discovery performance and reliability. Thus the registration of services needs to be subject to a lifetime, after which the registration expires. Service providers should periodically republish information to avoid deletion of previous published information.

Each independently running instance of an information service should keep aggregated data about all other instances. The benefit of this approach is that information can be retrieved by contacting any of the nodes, without bothering about the location from where the information regarding the service was published. It also allows fast retrieval of service data. Although this offers certain disadvantages in terms of loads on individual nodes the performance gains in term of access can overcome this cost.

3. Architecture

The architectural components of our discovery service can be divided into three main layers: the service interface, the wrapper and the repository. The top layer is the service interface which defines different attributes for describing a service and also methods for retrieving information from the service repository. The bottom layer (the repository) provides different variants for persistent storage of service related data. To provide an interface between different kinds of storage systems and our service interface, we provided a middle layer called the wrapper.

The service interface provides a way to describe service information and also different methods for accessing and manipulating service data. The choice of parameters was made to have more flexibility in providing service information. The list of parameters is given in Table 1 with a list of methods in Table 2.

Name	Description
Endpoint list {uri, encoding (soap, xmlrpc)}	Information on how to access the service including protocol and URL
Name	Name of the service
Admin Email	Email of the administrator who is managing the server
VO	The virtual organization of which this server is a part.
Site	Name of the site
WSDL url	URL for retrieving WSDL url
Provider_dn	Distinguished name of the provider of this service
Item {key , value}	Arbitrary key-value pair for describing the properties of the services

Table 1. Service description

Repositories provide a mechanism for the persistent storage of service related data. Different sets of repositories provide more or less flexibility for storage components since each of the repositories is suitable for different types of Grid environments. Relational databases are more suitable in a more stable environment with large numbers of services, where high availability is required.

On the other hand if the Grid environment is dynamic with reduced service availability or in cases where a lower response time is desired, then in-memory caches are best suited as storage repositories. One severe limitation of in-memory caches is the need for large amounts of RAM. In-memory caches do not scale well to large Grid environments with large numbers of services. An alternative is to use UDDI (Universal Description Discovery and Integration) - a more standardized approach for service description and discovery.

Interface method	Description
register	Publishes service information with the discovery service.
deregister	Removes the service information from the repository
find	Allow users to query the information service for particular service or services
find_key	Users can query on arbitrary key value pair(s) with this method.
find_server	Same as find but only provides the information about existing servers.

Table 2. Service access methods

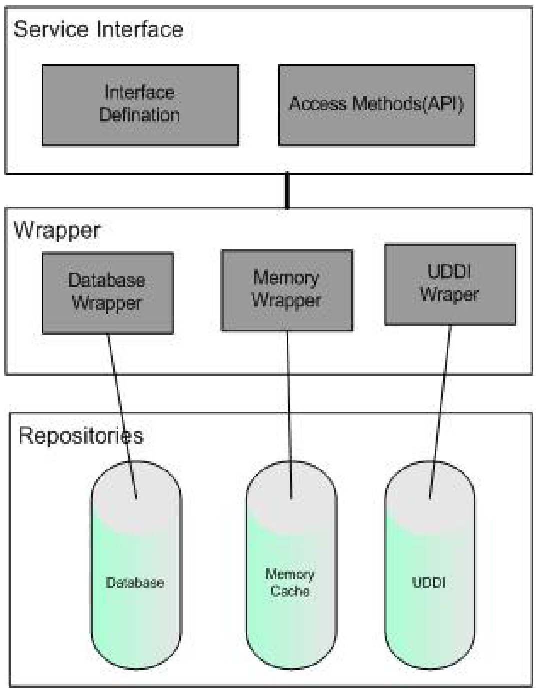


Figure 2. Discovery Service Architecture

The use of the abovementioned diverse storage systems leads to the requirement for a bridge to bind the variety of interfaces together. This wrapper layer is responsible for the seamless translation of requests from the interface of the DS to the underlying repository.

Another important aspect of the discovery or information service is the ability to co-exist with other instances. This is achieved through the replication of service data over all other nodes of network. We opted for a peer-to-peer method for dissemination of information using JINI [9] and MonALISA [10].

MonALISA was selected for information dissemination for the following reasons:

1. It was already used by various groups for collecting and aggregating monitoring information.
2. The publish/subscribe pattern for publishing and retrieving any arbitrary information in MonALISA allows us to follow OGSA [4] based standards.
3. MonALISA is already a core component of the Ultralight [11] project in which the Discovery Service is being used.

4. Implementation

A discovery service has been implemented as a web service within Clarens/JClarens. It is available in both Python (for Clarens) and Java (for JClarens [19]). The development work was carried out in three phases. Each phase built on the previous one. In this section we will give a description of each phase.

4.1. Phase I: Centralized Repository

In this phase the development of a prototype version (or skeleton) of our information service was undertaken. Initially we provided an API with the implementation of most of the methods mentioned in Table 2. Data on each service (parameters as presented in Table 1) was stored at a centralized location.

A discovery module was provided as a core component within the Clarens framework. Each instance of a Clarens server registered its services with the locally available information service. Replication was not possible between servers and consequently no communication was possible among different Clarens servers. Figure 3 shows this simple standalone discovery service.

4.2 Phase II: Service Replication and Distributed Discovery

Prior to Phase II there was no communication among individual instances of our discovery service. The support for the replication of the service data was built at this stage. We used JINI and MonALISA to achieve this purpose. MonALISA uses station servers which are dynamically interconnected using a JINI peer-to-peer network. These station servers are capable of collecting local monitoring data and of sharing it with other station servers or other interested clients. Arbitrary information can be sent to the MonALISA servers using ApMon packets – a library that uses simple XDR encoded UDP packets.

Figure 4 depicts how this was achieved. Each Clarens server upon its startup registered its services with the local repository and also pushed them to the MonALISA server. These service parameters were then published to any of the known MonALISA station servers. The station servers used an efficient agent-based system to propagate the service parameters to other discovery service instance. The MonALISA publish/subscribe model allowed its client to get interested information from its peer-to-peer network. A JINI-based client was embedded within each Clarens servers. This client received information from MonALISA and stored them in the local repository. In this way service information from one discovery instance was made available to others.

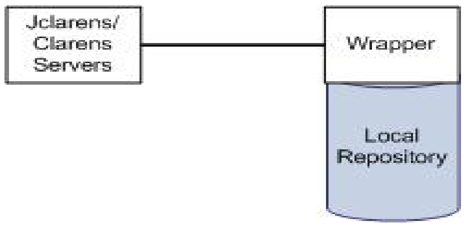


Figure 3. Standalone discovery architecture

The other development of note at this stage was to add service lifetime management. Keeping information for unavailable services not only decreases the efficiency of the system, but it may also hinder the overall working of the Grid systems. This hindrance is due to the fact that different Grid components rely on information obtained from the DS for their working. Thus lifetime management can be considered as a critical component of our DS. In our implementation of a discovery service, the publishing of service information is subject to a finite lifetime. Repositories are periodically purged to remove expired entries from the system. The service provider is responsible for renewing the service lifetime. The service expiry is managed by the provider of the service. So it can be easily adjusted based on the requirements and nature of the Grid applications.

4.3 Phase III: Interfacing with Standardized Components

A UDDI (Universal Description, Discovery and Integration) repository is a standardized component for describing, storing and retrieving service related data. But it is unsuitable for being deployed in large scale systems with dynamic nature of services. The use of UDDI repositories is mostly suitable in a stable environment – i.e. fewer services with high availabilities. We overcome this disadvantage of UDDI systems by using it as a repository with our discovery service. The built-in replication of the discovery service can help alleviate the deficiencies in UDDI use due to its stand-alone nature. Figure 5 shows how UDDI can take advantage of the global nature of our information service.

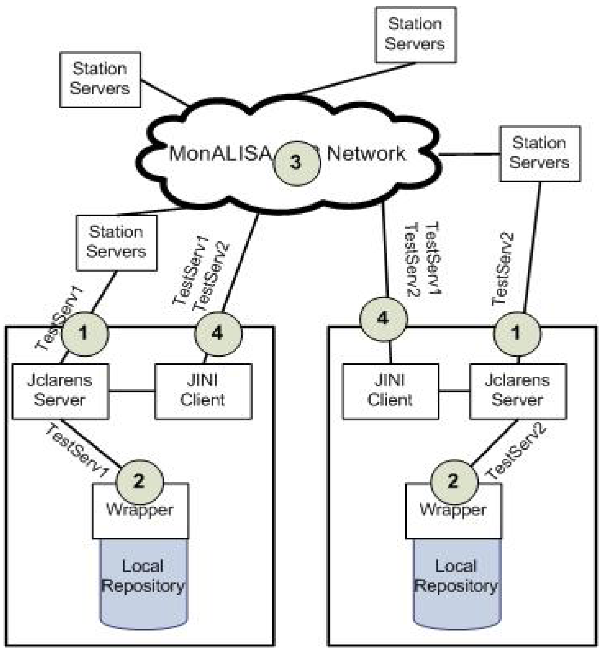


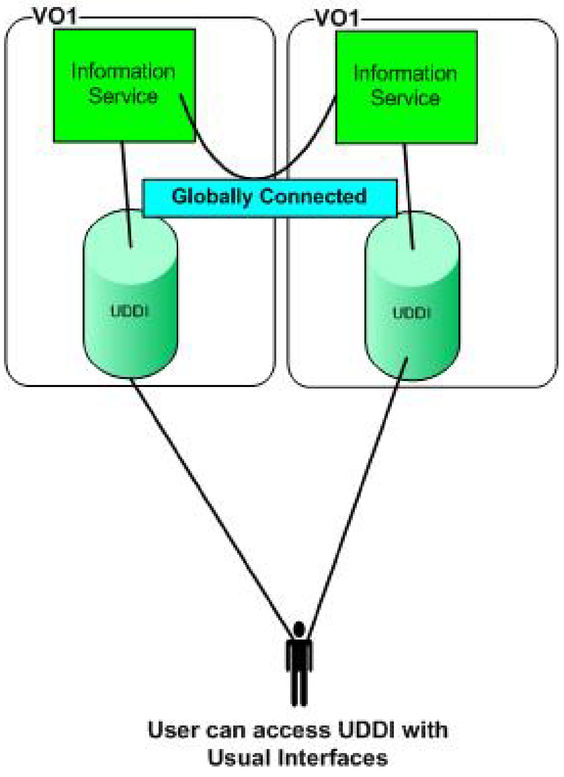
Figure 4. Service replication and distributed discovery

All distributed instances of UDDI are virtually linked to one another using the replication mechanism within our discovery service. The service data registered to UDDI running within one VO will be available to other VOs, thus making UDDI suitable to be deployed in large scale distributed systems.

A wrapper was built around the UDDI which translates each call made to our discovery service to the appropriate UDDI method call. With this wrapper it is possible to publish and query from the UDDI registry using both the Discovery Service interface and the standard UDDI interfaces, but the services added using the UDDI interface are not automatically propagated to other Discovery Service instances. This component is only available with the java implementation of Clarens. We used jUDDI [20] with uddi4j [21] for this purpose.

5. Discovery Service: Case Studies

The Discovery Service allows the service data that is published from anywhere, to be retrieved from anywhere in the Grid network. This functionality makes it suitable to be used as a global information index. In this section we present different applications as case studies in which our information system has served the above mentioned purpose.



FiFigure 5. Discovery Service as UDDI Repository

5.1 Job Monitoring

Job monitoring involves polling job related information such as job status within its lifecycle and to report its status back to user. Figure 6 presents a possible flow of information describing how job monitoring can be done using the discovery service. A user submits a job to the scheduler, which will schedule it to any of the execution clusters in any VO based on matchmaking. Now we are interested in knowing the status of the job during its execution. There may be many possible statuses of the job. A few of the possible status messages could be waiting, executing, halted, aborted etc. For the monitoring of the job we periodically publish the information of the job status to our discovery service. This information is published to any discovery node known to that site on which our job is being executed. Users can easily get the status information regarding the jobs submitted by them through any of the instances of the discovery service known to them. Even other services such as the job steering service can use such information for taking decisions like rescheduling the job to a more powerful site (in terms of resources).

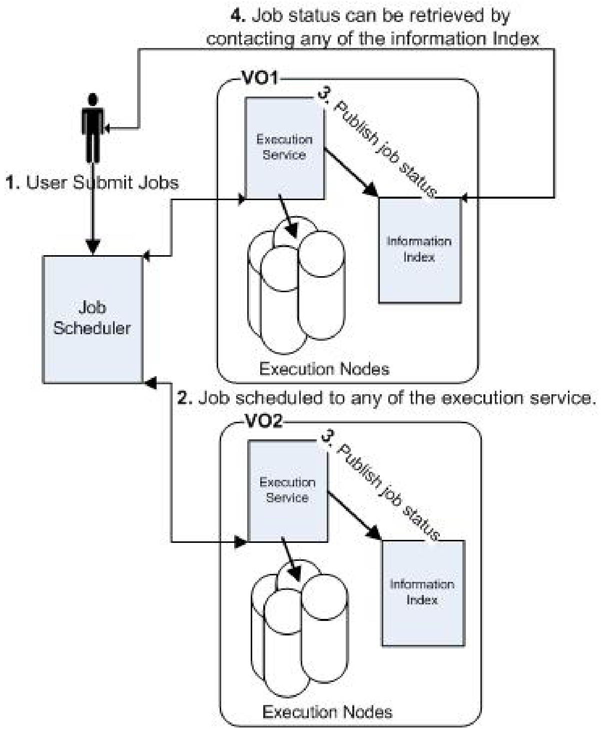


Figure 6. Discovery Service as Job Monitoring

5.2 Resource Management

A global information index can provide desirable functionality in a resource management framework. The resource related information is often required in the process of matchmaking i.e. selecting appropriate resources, in terms of CPU, memory, software requirements etc, for the execution of a job. When a user submits a job, its scheduling is carried out based on a number of parameters. These might involve the site policy, particular hardware and software requirements or other considerations. Our information index is used to discover Grid resources in order to facilitate the matchmaking process. Each Virtual Organization publishes information concerning the resources available at its disposal and the current status of their use.

Each site can publish information regarding the computing power it has into the global index; this involves the number of CPUs, the processing power available, the number of jobs being executed and the number of jobs in the queue. Whenever a user is interested in submitting a job, the job contacts the resource broker which does the matchmaking based on the information received from the information service. There are also other components that could be involved when scheduling a job such as maintaining user quota and carrying out accounting.

6. Performance Analysis

Results are included for demonstrative purposes and more accurate results are to follow in future work.

We calculated the time it takes for retrieving service related data from the information service using a java based XML-RPC client. The graphs in figure 7, 8 and 9 show the results of data retrieval with varying number of services. These results were calculated both from the memory-based and two different database-based storages. The service retrieval test does not include the time for making the connection or authenticating the user with the server. Apparently it seems that the exponential increase in service retrieval time as number of service increase is due to the overhead involved in parsing the XML-RPC response from the Clarens server. This could be further verified in the exhaustive testing of our system which is currently being done. We also noted that the in-memory cache is suitable for fast retrieval of services data. The only problem is with the memory over flow when we register a large number of services or data. Another test was to measure the time the information service takes to replicate service information to other instances of information services over the wide area network. The Service retrieval time was calculated based on the difference in time between a service being registered at one node of JClarens (running at NUST, Pakistan) and becoming available at another node (running at Caltech, USA).

The response we got varied from 3 seconds to 10 seconds, with it rarely going above 22 seconds. The mean of different observations was 10.7 ± 8.3 seconds. Therefore on average it took about 10.7 seconds for the dissemination of one service across the Clarens network through the MonALISA replication mechanism. Figure 10 presents the values obtained for service retrieval for different number of attempts. The variance in the upper values is attributed to the network latency prevalent in our network. Being initial results we did not try to find out the actual values of these latencies and their effect on the service retrieval data.

All of these tests were conducted on an Intel P4 machine with 2.8 GHz process with 256 MB memory.

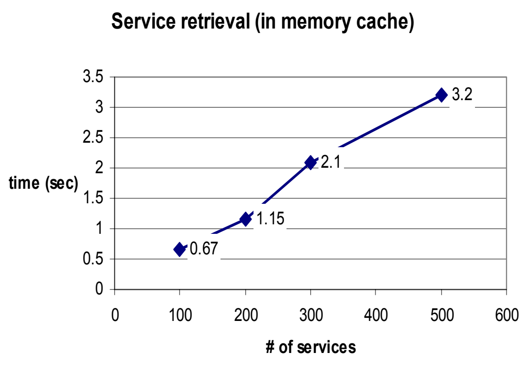


Figure 7. Service retrieval (in memory cache)

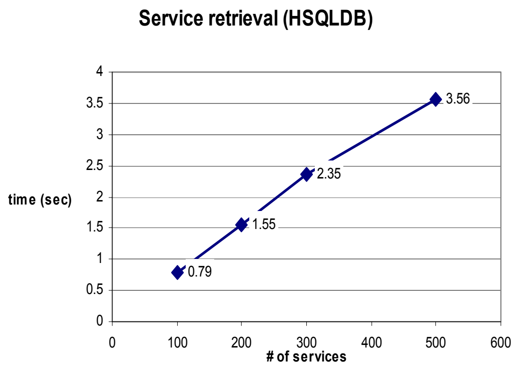


Figure 8. Service retrieval (HSQLDB)

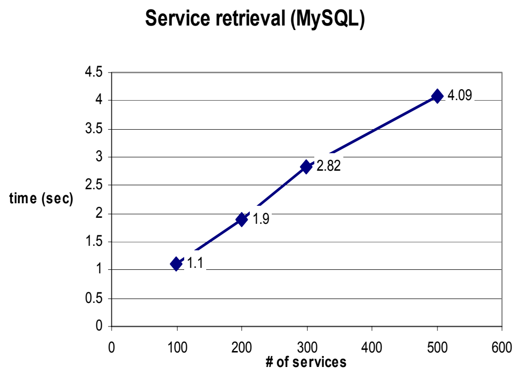


Figure 9. Service retrieval (MySQL)

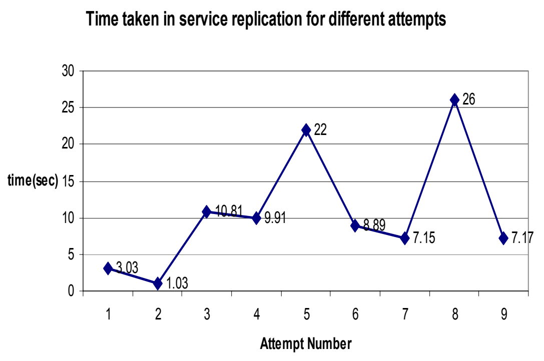


Figure 10. Service replication

7. Related Work

A Discovery System is a core component of any Grid implementation [22], [12] and there are many existing components available which can perform the task of discovering Grid resources and services. In this section we discuss some important developments in this area of research and provide a critical analysis of the strengths and weaknesses of each system..

The DNS (Domain Name System) is perhaps the best known example of a discovery/information system. DNSs are usually referred as internet directory services. They provide mappings between IP addresses and the internet domain names. DNS are however not suitable for use in the Grid environment because of their static nature and limited query interface.

MDS4 [12] is the discovery and monitoring component of the Globus [13] toolkit4. It is based on emerging Grid standards such as WSRF [15] and WS-Notification [16]. It comprises an index service, which collects and publishes aggregated information on Grid resources and a trigger service which collects information and performs certain actions based on the specified conditions. This information can be accessed using the WSRF query and subscription interfaces. MDS4 mainly focuses on collecting monitoring information by interfacing it with detailed monitoring systems but can also be used for the discovery of resources. Recently we have held discussions with the Globus group to provide a bridge between our Discovery Service and MDS. This will allow us to publish our DS data into MDS and pull service information from MDS. In this sense, MDS can be used as yet another backend repository for our DS.

RGMA [14] is a relational implementation of GMA [17] (Grid Monitoring Architecture), an architecture initially presented by the GGF [18] (Grid Global Forum). This monitoring and information system works like a large relational database with SQL queries to retrieve information. The architectural components *of* RGMA involve a virtual database and a set of producers and consumers. A virtual database is used to organize users' data into tables and allows data manipulation using SQL queries. Each virtual database in a Grid will have a unique name and will belong to any members of the virtual organization. Information is queried from these databases by contacting the consumer and information is provided to databases by contacting the producers. A mediator or centralized registry holds information about the currently available producers. This centralized registry is a bottleneck in the presence of a large number of producers. Moreover, services are not published and retrieved instantly and dynamically since replication of the data is not in real time.

8. Conclusion

This paper describes the design, implementation and performance analysis of our discovery and information system. We presented how our system tackles the most demanding issues of service registration, information propagation and service discovery and access in widely distributed and diverse systems infrastructures. We discussed some unique features of the system - the most important of which include interfacing with different backend systems - which incorporates flexibility, avoiding stale service data by imposing a lifetime for each service entry and providing an updated view of the dynamically varying distributed systems. The performance analysis is presented to allow readers to gauge how our system behaves with changing number of services. We also presented ongoing work on the case studies which shows the usability of our system as a main facilitator for other components of grid such as job and network monitoring, scheduling and resource management.

The support of our organizations and of the CMS experiment at CERN is gratefully acknowledged..

References

[1] F. van Lingen et al.,, "The Clarens Web Service Framework for Distributed Scientific Analysis in Grid Projects", In Proceedings of the International Conference on Parallel Processing Workshops, Oslo, Norway, June 2005, ISBN 0-7695-2381-1, pp45-52. See also: http://clarens.sourceforge.net/

[2] Open Science Grid http://www.opensciencegrid.org

[3] I Foster, A. Iamnitchi, "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing". Lecture Notes in Computer Science, 2003. Springer Verlag

[4] Karan Bhatia. "Peer-To-Peer Requirements On The Open Grid Services Architecture Framework" GFD-I.049 OGSAP2P Research Group.

[5]"The Napster Homepage" – http://www.napster.com.mars

[6] "Gnutella" - http://www.gnutella.com/

[7] "SETI at HOME" - http://setiathome.berkeley.edu/.

[8] UDDI - http://www.uddi.org/

[9] Jini.org - The Community Resource for Jini Technology - http://www.jini.org/

[10] I. Legrand et al "MonALISA: an Agent Based, Dynamic Service System to Monitor, Control and Optimize Grid Based Applications", in Proceedings of Computing for High Energy Physics, Interlaken, Switzerland, 2004.

[11] H. Newman et al.,, , "The Ultralight project: The Network as an Integrated and Managed Resource for Data Intensive Science", in Computing In Science and Engineering, Issue on grid computing, 2005

[12] JM Schopf, et al.,, "Monitoring and Discovery in a Web Services Framework: Functionality and Performance of the Globus Toolkit's MDS" Argonne National Laboratory Technical Report# ANL/MCS-P1248-0405

[13] The Globus Alliance - http://www.globus.org/

[14] AW Cooke et al.,"The Relational Grid Monitoring Architecture: Mediating Information about the Grid"- Journal of Grid Computing (2004) 2: 323–339 © Springer 2005

[15] The WS-Resource Framework (Specifications), http://www.globus.org/wsrf/specs/ws-wsrf.pdf

[16] Web Services based Notification (WS-Notification), ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BaseN.pdf

[17] GMA – 'A Grid Monitoring Architecture' GGF document available at: http://www-didc.lbl.gov/GGF-PERF/GMA-WG/

[18] GGF – Global Grid Forum , http://www.gridforum.org/

[19] M. Thomas et al., "JClarens: A Java Framework for Developing and Deploying Web Services for Grid Computing", In Proceedings of 2005 International Conference on Web Services (ICWS 2005), Orlando, Florida, July 11-15, 2005, ISBN 0-7695-2409-5 pp141-148

[20] jUDDI - http://ws.apache.org/juddi/

[21] UDDI4J - http://ws.apache.org/juddi/

[22] K Czajkowski et al., "Grid Information Services for Distributed Resource Sharing" IEEE Int. Symp High Performance Distributed Computing Proc (2001). pp. 181-194. [23] "National Virtual Observatory" - http://www.us-vo.org/index.cfm

[24] *S. Andreozzi et. al. "*GridICE: a Monitoring Service for Grid Systems". In Future Generation Computer Systems Journal, Elsevier, 21(4):559-571, 2005